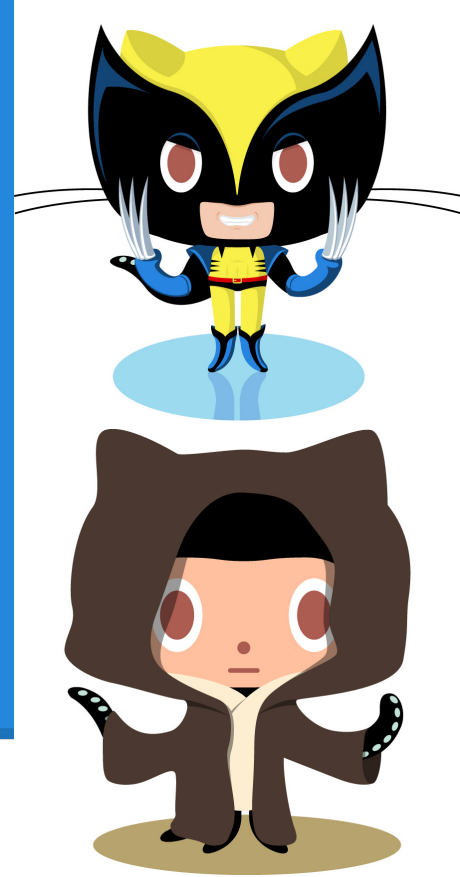


Git Internals Continued

by Jonathan Miedel and Alvin Wang
Shannon Lee



Last Time on 174

- Git Internals
- Content Addressable File System
- Hashing (specifically SHA-1)
- Git Objects
- Packfiles

Quiz

1. What is the utility to learning git internals?
2. What is the difference between plumbing and porcelain commands?
3. What does content-addressable filesystem mean? How is it different from path based file-systems.
4. What are the 3 pieces of information that you can find in any git object?
5. Give 4 types of git objects.
6. If I ran `git hash-object -w` on a file and it gave me back a hash “aabbaabb...”, what is the path relative to the top level repository folder this hashed-object is located in?
7. What are loose objects?
8. What command in Git can be used to pack objects? And describe the packing process.
9. What type of git objects can be found in a tree object?
10. What information can be found in the packfile index?
11. What command can be used to inspect a tree object with the hash “a4v2c4”

Refs

- References
- How Git refers to commits
- `.git/refs`
 - Branch information

Special Refs

- HEAD – The currently checked-out commit/branch.
- FETCH_HEAD – The most recently fetched branch from a remote repo.
- ORIG_HEAD – A backup reference to HEAD before drastic changes to it.
- MERGE_HEAD – The commit(s) that you're merging into the current branch with `git merge`.
- CHERRY_PICK_HEAD – The commit that you're cherry-picking.

The Refspec

- The refspec defines the **mapping** between local branch references and remote branch references when working with remotes
- You define a refspec for each remote (not each branch)
- The refspec can be found in the `.git/config` file
- The format for the refspec is
- `<source>:<destination>`
 - source refers to the remote source and destination refers to the destination of the references locally
 - The `+` tells git to update the references even if they are not fast-forward updates, it is optional

```
[remote "origin"]
```

```
url = git@github.com:Dogfalo/materialize.git
```

```
fetch = +refs/heads/*:refs/remotes/origin/*
```

Throwback to Exam Extra Credit

- EC5. Given there is a branch `origin/toBeDeleted`, write the command that would delete the `toBeDeleted` branch off of the remote.
- Now that we know the refspec is formatted as `<source>:<destination>`, it makes much more sense that `git push origin :toBeDeleted` deletes a remote branch.
- What is this doing?
- Remember that Git likes to keep things around for a while before actually deleting them
- We are updating remote branch reference with nothing as the source so that the local branch reference

Refspec Continued

- Continuing on the example, when we run:
 - `git remote add origin https://github.com/Dogfalo/materialize`
- Git goes to the server and fetches down all references under `refs/heads/` and places them in our local repository in `refs/remotes/origin`
- You now have all the references related to that remote!

Refspec Cont2.

- We've been using shorted refs all along
- branch "test" is short for "refs/heads/test"
- tag v1.5 is short for "refs/tags/v1.5"
- origin/master is short for "refs/remotes/origin/master"
- The HEAD file in refs/remotes/origin/ does not indicate the location of your current HEAD, but rather it defines the default branch of the

Packed Refs

- git gc, which we saw in the past packs our loose objects, also packs our references up
- The packed refs are stored in .git/packed-refs as a file

```
# pack-refs with: peeled fully-peeled
70a6b391c55bf7a390ccbf3f9e9be02d6a71474a refs/heads/cards
7e430b3c979603fda978cdf1816202ff35b56de3 refs/heads/checkbox
9f850e4d896a8ba6decc6d1a5eee708dd48af57d refs/heads/dropdown
f1b961f8faee15383ea7b4f4665187f254150cf6 refs/heads/gh-backup
f1130d8676f51a77c3bb3eb7b5beca23ea4baefd refs/heads/gh-pages
f1130d8676f51a77c3bb3eb7b5beca23ea4baefd refs/heads/master
ce8f6f3a42b17f779784e81ac8f7240504710018 refs/heads/materialbox
e5f0dc18d0e50f4766caa0cf355f480a6ba9ad00 refs/heads/parallax
c69481ebdfffa192fd24350e00c5cbfec9bfaf90d refs/heads/progress
59292c0d2323edaaae80b1a3a2642792b1d5e454 refs/heads/radio
94f8ae2be72498e075a9437e878c4f31fbd992f3 refs/remotes/origin/aboutus
e1b9967e358d20867db8a3c98b1a4f5046593447 refs/remotes/origin/buttons
7e430b3c979603fda978cdf1816202ff35b56de3 refs/remotes/origin/checkbox
df076ca972cbc26e012c224444fa0387551e8efc refs/remotes/origin/develop-alex
9f850e4d896a8ba6decc6d1a5eee708dd48af57d refs/remotes/origin/dropdown
f1b961f8faee15383ea7b4f4665187f254150cf6 refs/remotes/origin/gh-backup
f1130d8676f51a77c3bb3eb7b5beca23ea4baefd refs/remotes/origin/gh-pages
d30d6a8acb555e641edb216647e5a082c375a0ca refs/remotes/origin/grid
f1130d8676f51a77c3bb3eb7b5beca23ea4baefd refs/remotes/origin/master
ce8f6f3a42b17f779784e81ac8f7240504710018 refs/remotes/origin/materialbox
ddeb72b59fd9ae7320fdb182f4572a43625efb70 refs/remotes/origin/navbar
e5f0dc18d0e50f4766caa0cf355f480a6ba9ad00 refs/remotes/origin/parallax
59292c0d2323edaaae80b1a3a2642792b1d5e454 refs/remotes/origin/radio
926d6663429002a78d316ab8a0121a213a9915ac refs/remotes/origin/shadows
11f58f651c387e5771b3a2702d8a3b81722d4fe6 refs/remotes/origin/table
71cc1bbf39b80f86b2c54fd5aa3d0a349ecc0cb3 refs/stash
```

Basic Data recovery

What are some ways we can “lose” our commits in git?

- `git reset --hard HEAD~3`
- `git branch -D topicBranch`

Process of undoing git reset --hard

- Current structure is A<--B<--C<--D
- git reset --hard HEAD~2 puts us at A<--B
- To restore back to A<--B<--C<--D, we just need the references to the commits
- They are no longer in the log, but now are in reflog (located in .git/logs/refs)
- The reflog stores changes to the HEAD as a queue
 - hashB HEAD@{0}: reset moving to B
 - hashD HEAD@{1}: commit added D
 - hashC HEAD@{2}: commit added C

Reflog example output

```
Alvins-MacBook-Pro:materialize Alvin$ git reflog
f1130d8 HEAD@{0}: checkout: moving from gh-pages to master
f1130d8 HEAD@{1}: merge master: Fast-forward
abf5f48 HEAD@{2}: checkout: moving from master to gh-pages
f1130d8 HEAD@{3}: commit: updated to new logo
51006ef HEAD@{4}: pull: Fast-forward
abf5f48 HEAD@{5}: checkout: moving from gh-pages to master
abf5f48 HEAD@{6}: merge master: Fast-forward
20668a8 HEAD@{7}: checkout: moving from master to gh-pages
abf5f48 HEAD@{8}: commit: changed logo positioning
20668a8 HEAD@{9}: checkout: moving from gh-pages to master
```

Recovering continued

- You want to grab the latest commit of what you want to recover, in this case hashD
- right now these are dangling commits because they do not reside in a branch and will eventually be deleted
- to save them:
- `git branch recovery hashD`
- This creates a new branch called recovery which contains the commits of hashD and everything before
- Now we can just merge with master

Recovery scenario 2

- That was fairly easy, but what if we don't have the commit in the reflog?
- This may happen because the reflog is not copied to anything during git push; after a git clone, you have no reflog
- However remember that the reflog, is still just a reference in the end, this means our data may still be there floating around.
- Now we have to run `git fsck --full`, to show objects that aren't being pointed to by anything else.

Recovery 2 Continued

```
Alvins-MacBook-Pro:materialize Alvin$ git fsck --full
Checking object directories: 100% (256/256), done.
Checking objects: 100% (2075/2075), done.
dangling commit dde2d26cfcece4dcb531c4db564ea838c0068655
dangling commit bc838687937d349932c74345cd0cbd9b4009e366
dangling blob 3b465124cbc5af77946a123e0edf8353749af60d
dangling blob d6a8153734ecf60da2281fd0d045728b19465bea
dangling commit 8bcabbab07792dc826b8d9aa9abb2e5ec25de976
dangling blob 97aac5ab286248023002ac3966db2b3426c50ed0
dangling commit 3d3201ca8a2cba37877db3c6a8113e74aa012af1
dangling commit 699461234bcc4cb5c18996966884e34b0e4f4ba2
dangling commit 365534e567b9f4ba29a77d7c2ae8d5428aa3b794
dangling blob cb1854314dad70470587b451cbe867f4b92c2d59
```

- Here we use the skills/commands we learned last class. `git cat-file -p <hash>` will show us what is in these dangling commits.
- Then we just `git branch recovery <hash>` just as before
- N.B. “recovery” is just a name which can be changed to whatever you want

Transfer Protocols

- Used for network actions
 - git pull, git clone, git push
- Dumb Protocol
 - Mostly outdated
 - Used for reading only
- Smart Protocol
 - Supported by most git hosting services

The Dumb Protocol

- Uses simple HTTP
- Does not require any git specific code
- Series of GET requests to a known file structure to receive all necessary information
- Does not provide support from transferring from client to server
- Requires minor set up on server side

Implementation

- GET info/refs
 - returns list of remote references
- GET HEAD
 - returns the head reference
- GET objects/...
 - start fetching objects
 - uncompressed and check to see what else is needed
 - if you 404 request the pack

Smart Protocol

- More efficient
- Allows for reading and writing
- Uses HTTP/HTTPS or SSH
- More secure (if using HTTPS or SSH)

Smart Protocol SSH Upload

- Runs git send-pack
 - Sets up ssh connection to server
 - runs git receive-pack on server
- The server then sends back a line for each reference
- Based on this the client determines what it has that the server does not

Smart Protocol SSH Upload Continued

- Then the client sends a line back for each reference it was missing
- The server then returns if it was ok
- Next the client sends the packfile

Smart Protocol SSH Download

- Runs git fetch-pack
 - Sets up ssh connection to server
 - runs git upload-pack on server
- Server responds with all of refs plus a HEAD
 - the head is used to tell the client what to check out if the command is a clone
- The client then sends back what it wants and has to the server

Smart Protocol SSH Download Cont.

- The server replies with an appropriate pak file for the requested data

Smart Protocol HTTP/HTTPS Upload

- GET \$GIT_URL/info/refs?service=git-receive-pack
 - works the same way as before
- Initiates a new connection with data from git upload-pack
 - provides the pack and references unlike before

Smart Protocol HTTP/HTTPS Download

- GET `$GIT_URL/info/refs?service=git-upload-pack`
 - works the same was as the other download
- POST `$GIT_URL/git-upload-pack HTTP/1.0`
 - initiates a new connection
 - response has success or failure along with packfile

Example of Server Responding

005bca82a6dff817ec66f4437202690a93763949

refs/heads/master report-status \

delete-refs side-band-64k quiet ofs-delta \

agent=git/2:2.1.1+github-607-gfba4028 delete-refs

003e085bb3bcb608e1e84b2432f8ecbe6306e7e7 refs/heads/topic

0000

Explanation of Response

- First 2 bytes are the length of the line
- First line is a list of capabilities
- Each following line is a ref and its hash

Explanation of client response

- First 2 bytes are the length of the line
- First line has capabilities
- Each line has old hash and new hash for the ref

Advantages and Disadvantages of SSH

- Easy server side
- Secure
- No anonymous access
 - bad for open source
- Harder for users
 - Sometimes firewalled
 - Have to generate key

Next Week in Git Stuco

- git filter-branch
- git bisect
- Hooks

Homework

Due next Saturday 11:59PM

Will be released on Saturday

Will be give a .zip file, ref log will be deleted